Early Evaluation of the IBM p690 *

P. H. Worley †

T. H. Dunigan, Jr. ‡

M. R. Fahey §

J. B. White III ¶

A. S. Bland ‖

Abstract

Oak Ridge National Laboratory recently received 27 32-way IBM pSeries 690 SMP nodes. In this paper, we describe our initial evaluation of the p690 architecture, focusing on the performance of benchmarks and applications that are representative of the expected production workload.

1 Introduction

Computational requirements for many large-scale simulations and ensemble studies of vital interest to the U.S. Department of Energy (DOE) exceed what is currently offered by any U.S. computer vendor. Examples are numerous, ranging from global change research to combustion to informatics. It is incumbent on DOE to be aware of the performance of new or beta systems from high-performance-computing vendors that will determine the performance of future production-class offerings. It is equally important that DOE work with vendors in finding solutions that will fulfill DOE's computational requirements.

In support of this mission, Oak Ridge National Laboratory (ORNL) is currently performing in-depth evaluations of a number of high-performance computer systems. A cluster of IBM pSeries 690 (p690) systems is the most recent subject of the early-evaluation project.

ORNL received its first p690, a 16-processor SMP node, in October, 2001. Another 23 nodes arrived in January of 2002, and the original node received an upgrade, such that all systems now have 32 processors. Another 3 nodes arrived in April. In May, all nodes were interconnected with two "planes" of IBM SP Switch2. As the switch was just being installed at the time of these experiments, we discuss single node performance only.

*This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Batelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

†Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367 (WorleyPH@ornl.gov)

‡Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367 (DuniganTHJr@ornl.gov)

§Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 4500N, Oak Ridge, TN 37831-6203 (FaheyMR@ornl.gov)

¶Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 4500N, Oak Ridge, TN 37831-6203 (WhiteJBIII@ornl.gov)

‖Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 4500N, Oak Ridge, TN 37831-6203 (BlandAS@ornl.gov)

0-7695-1524-X/02 $17.00 (c) 2002 IEEE

The primary tasks of the evaluation are to:

- determine the most effective approaches to using the architecture;
- evaluate benchmark and application performance, and compare with other systems;
- evaluate system and system-administration software reliability and performance;
- predict scalability, in terms of both problem size and number of processors.

The evaluation results in this paper address the first two tasks. In particular, we compare the performance of codes on the p690 with that on previous generation systems from IBM, Compaq, and Cray, to indicate what performance users might expect when porting their codes to this new platform. While standard benchmarks are used when appropriate, and for comparison with other evaluations, the emphasis here is on application-relevant studies for applications of importance to DOE. The applications involved in our current evaluations include codes drawn from astrophysics, climate modeling, and fusion projects in the DOE *Scientfic Discovery Through Advanced Computing* (SciDAC) program [10].

2 System

We describe the performance of single IBM pSeries 690 symmetric multiprocessor (SMP) nodes running IBM AIX 5.1, where each node has 32 1.3GHz POWER4 processors. Each processor has two floating-point units, and each unit can produce a multiply-and-add result per clock cycle in pipelined operation. Therefore, each processor can complete four floating-point operations per cycle. The pipeline for each unit is six cycles, up from the three-to-four-cycle pipeline in the POWER3.

Each POWER4 chip has two processors, so each node has 16 chips. The two processors on a chip share a single L2 cache for data and instructions, but each processor has its own private L1 data cache and L1 instruction cache. The L1 data caches are 32KB in size and two-way set associative, with a first-in-first-out (FIFO) replacement policy. The FIFO replacement policy can make blocking for the L1 cache unproductive because it limits reuse; data loaded earlier are overwritten even if they were used more recently. The shared L2 on each chip is 1440KB in size. Sharing the L2 cache may be a disadvantage when running independent processes on each processor, but it may be an advantage when running threads that share an address space. "False sharing", separate processors accessing different data on the same cache line, may improve instead of degrade performance.

Four POWER4 chips (eight processors) are connected to form a multi-chip module (MCM), and the L2 cache on each MCM chip has point-to-point access to the L2 caches on the other three. Each MCM goes through four 128MB L3 caches to memory, where each L3 cache connects to a separate memory controller. Therefore, each L3 cache is associated with a certain region of real memory, not a certain processor. Memory pages are striped by cache line across the four L3 caches and memory controllers, maximizing the cache size and memory bandwidth available to any single processor.

Each memory controller has one or two ports to memory, depending on the size of the memory. For the results presented here, we used nodes with 32GB memory and one port per controller. Nodes with 64GB and 128 GB have two ports. Because one port is almost enough to saturate the bandwidth of a memory controller, two ports do not double the effective bandwidth. Our benchmarks show no significant performance variation between running on 32GB, 64GB, and 128GB nodes.

In addition to caching memory that is local to the MCM, the L3s can cache references to memory on other MCMs. On a 32-processor p690 node, four parallel memory busses connect four MCMs in a ring, one bus for each POWER4 chip in an MCM. Memory requests from an L3 on one MCM can be met by memory, L3, or L2 on another MCM.

The performance of prefetching on the p690 is affected by the configuration of the L3 caches. The line size for the L1 and L2 caches is 128B, but it is 512B for the L3s, and the L3s can prefetch a full 512B line or not based on boot-time configuration. For the experiments described here, 512B L3 prefetch was enabled, resulting in a 10%-15% performance improvement over previous benchmark results.

3 Methodology

This is an early-evaluation study, limited by the simultaneous evolution of the system. As such, the focus has been as much on identifying and reporting performance problems as it has been on performance characterization. The approach taken has been to:

- port codes and kernels representative of the expected work load;
- collect benchmark data, allowing comparison with other platforms;
- run the codes in a variety of ways, attempting to identify and characterize the unique features of the architecture.

Our application codes are the following.

1) AORSA3D. The AORSA3D code solves for the wave electric field and heating in a 3-D stellerator plasma heated by radio-frequency waves using an all-orders spectral algorithm[7]. It represents an important application in the "Numerical Computation of Wave-Plasma Interactions in Multi-dimensional Systems" DOE SciDAC project. AORSA3D has three major computational phases:
   - matrix generation,
   - complex linear system solution, and
   - current calculation.

   AORSA3D uses SCALAPACK[2] to solve the linear systems.

2) EVH1. The Enhanced Virginia Hydrodynamics #1 (EVH1) code represents an important application kernel in the "TeraScale Simulations of Neutrino-Driven Supernovae and Their Nucleosynthesis" DOE SciDAC project. EVH1 is based on VH1[3], a multidimensional ideal compressible hydrodynamics code written by John Blondin at North Carolina State University. EVH1 uses the Lagrangian remap version of the Piecewise Parabolic Method (PPM) of Colella and Woodward[4]. In particular, EVH1 uses second-order operator splitting and 1D Lagrangian hydrodynamics in each coordinate direction. In the parallel implementation, the domain decomposition is restructured as each coordinate direction is treated, to keep the given direction on processor.

3) PCTM. The Parallel Climate Transitional Model (PCTM) is the next generation of the Parallel Climate Model[11, 1]. It is made up of atmosphere, ocean, land-surface, and sea-ice component models, and a coupler to exchange fluxes between the component models. The atmospheric model is a recent version of the Community Climate Model (CCM)[9], developed at the National Center for Atmospheric Research (NCAR). The ocean model is POP (Parallel Ocean Program)[8], developed at Los Alamos National Laboratory (LANL), the National Physical Laboratory (NPL), and NCAR.

4) CCM/MP-2D. CCM/MP-2D is the massively parallel implementation of version 3.6.6 of the CCM [6]. It was developed originally to determine how best to parallelize the CCM, and the results from this research are being used in the parallelization of the Community Atmospheric Model (CAM), the successor to the CCM. CCM/MP-2D is currently used for benchmarking parallel systems.

Our kernel codes are the following.

5) Linpack. The High-Performance Linpack (HPL) benchmark solves a (random) dense linear system of equations in double-precision arithmetic on distributed-memory computers [5]. HPL solves the linear system by computing the LU factorization with partial row pivoting of the $n \times (n + 1)$ coefficient matrix $[A\ b]$. The data are block-cyclicly distributed onto a two-dimensional $P \times Q$ grid of processors to ensure load balancing. The right-looking variant of the LU factorization is used. The HPL code contains many possible variants for various operations. It is left up to the user to experimentally find the optimal set of parameters for a given machine configuration.

6) PSTSWM. The Parallel Spectral Transform Shallow Water Model (PSTSWM) represents an important computational kernel in spectral global atmospheric models [12, 13]. As 99% of the floating-point operations are multiply or add, it runs well on systems optimized for these operations. PSTSWM exhibits little reuse of operands as it sweeps through the field arrays; thus it exercises the memory subsystem as the problem size is scaled and can be used to evaluate the impact of memory contention in SMP nodes.

7) PCRM. The Parallel Column Radiation Model (PCRM) is a slightly modified version of the column radiation model used in version 3.6.6 of the CCM. It also represents an important computational kernel in atmospheric models. It differs significantly from PSTSWM in that over 6% of the floating-point operations are divide or square root, and it exhibits much better operand reuse. Therefore, it makes vastly different demands on the processor and compiler.

8) COMMTEST. COMMTEST is a suite of codes that measure the performance of MPI interprocessor communication. COMMTEST differs somewhat from other MPI benchmark suites in its focus on determining the performance impact of communication protocol and packet size in the context of "common usage". However, the performance we report here should be similar to that measured using other interprocessor communication benchmarks.

All parallel codes use MPI for interprocess communication.

We compare benchmark results with performance measured on one or more of the following platforms.

- Compaq AlphaServer SC at Pittsburgh Supercomputing Center: 750 ES45 4-way SMP nodes and a Quadrics QsNet interconnect. Each node has two interconnect interfaces. The processors are the 1GHz Alpha 21264 (EV68).
- Compaq AlphaServer SC at ORNL: 64 ES40 4-way SMP nodes and a Quadrics QsNet interconnect. Each node has one interconnect interface. The processors are the 667MHz Alpha 21264a (EV67).
- IBM SP at the National Energy Research Supercomputer Center (NERSC): 184 Nighthawk II 16-way SMP nodes and an SP Switch2. Each node has two interconnect interfaces. The processors are the 375MHz POWER3-II.
- IBM SP at ORNL: 176 Winterhawk II 4-way SMP nodes and an SP Switch. Each node has one interconnect interface. The processors are the 375MHz POWER3-II.
- Cray Research T3E at NERSC: 644 Alpha 21164 (EV5) processors running at 450 MHz.

Table 1 summarizes the peak processor performance and measured point-to-point MPI latency and bandwidth on these platforms.

	Peak	intranode MPI		internode MPI	
System	GFlop/s/proc.	latency (μs)	BW (GBytes/s)	latency (μs)	BW (GBytes/s)
IBM p690	5.2	3.0	2.0	-	-
Compaq (ES45)	2.0	5.0	.73	5.0	.27
Compaq (ES40)	1.3	5.8	.66	5.5	.20
IBM SP (ORNL)	1.5	6.6	.45	16	.14
IBM SP (NERSC)	1.5	8.6	.60	17	.38
T3E-900	0.9	-	-	11	.34

Table 1: Performance Characteristics of Parallel Systems

On each system, we use the vendor-supplied MPI, hopefully representing the best MPI performance. In all of our IBM experiments, the "`MP_SHARED_MEMORY`" environment variable was set to "`yes`", allowing the MPI implementation to use shared-memory communication between processes in the same SMP node.

In addition to benchmarking, experiments are presented looking at issues of memory contention, the impact of process placement on performance, and the impact of space sharing a single node.

4 Results

4.1 Application Benchmarks

Benchmarks results for the four application benchmarks are described in Figures 1-4. Remember that the p690 results are all within a single SMP node, while the Compaq and ORNL SP results include internode communication when using more than 4 processors.

While not a performance result per se, one of the most important results of our evaluation of the p690 is that the applications included in this study all "just work", even using binaries from POWER3 systems. Recompiling and targeting the POWER4 processor also works and provides some modest performance improvements. In the performance results described below, all codes were compiled on and for the p690.

AORSA3D. AORSA3D is typically run in a scale-up mode. The number of Fourier modes retained by the model is increased as the number of processors is increased, subject to keeping the memory size per process approximately constant. Practical considerations restrict the choice of Fourier modes, and not all mode counts are acceptable. For comparison between platforms, a maximum memory size per process was determined on the ORNL SP and used on the other platforms. Similar per-process memory requirements apply to the other systems, so this does not significantly skew the results.

The experimental results graphed in Figure 1 describe the performance in terms of the ratio of the number of modes to the execution time. If M is the number of modes, then the total memory requirement is $O(M^2)$, while the computational complexity contains both $O(M^2)$ and $O(M^3)$ terms. Thus the complexity for a fixed per-process memory size increases as a function of M. This eventually leads to a decrease in the number of modes-per-second execution rate as M increases, independent of any parallel overhead.

The p690 performance is between 2.2 and 2.7 times better than the SP system, with the advantage increasing for larger numbers of processors. The clock ratio (1.3GHz/375MHz) is 3.5, and the p690 is getting 63%-77% of the clock improvement.

EVH1. The EVH1 results in Figure 2 are for a fixed-size problem: Sedov-Taylor blast-wave solution in a 2D spherical geometry using a 128×128 computation grid. The p690 shows excellent scalability up to 32 processors for this problem size (as do the IBM SP and Compaq systems). The p690 per-processor performance is 2.4-3.2 times better than that of the SP, achieving 69%-91% of the clock ratio between the two systems.

PCTM. The PCTM benchmark results in Figure 3 are for a fixed-size problem. The atmosphere was run using a $128 \times 64 \times 18$ computational grid, and the ocean was run using a $388 \times 288 \times 40$ computational grid. The results are plotted in terms of the number of model years that can be simulated in one real-time day as a function of the number of processors. Efficiency (i.e., $E_p = T_1/(p * T_p)$) is greater than 60% for up to 32 processors on all platforms. The p690 performance is 2.7-3.0 times that of the SP, which is 75%-86% of the clock ratio.

CCM/MP-2D. The CCM/MP-2D benchmark results are for a fixed-size problem, T42L18, representing a $128 \times 64 \times 18$ computational grid. The graph in Figure 4 describes the computational rate as a function of processor count. The flop count was determined using SpeedShop on an SGI Origin. As this count gives equal weight to all floating-point operations (including, for example, square root), its meaning in terms of absolute performance is somewhat suspect. The graph is best interpreted as a consistently weighted plot of inverse runtime. The efficiency on the p690 when using 32 processors is 56%, and the performance is 2.3-2.7 times that of the SP. This represents 66%-77% of the clock ratio.

CCM/MP-2D includes instrumentation and performance-data analysis tools, allowing us to identify the sources of performance loss. The graph in Figure 5 describes these sources on the p690 as a function of the number of processors. The sources are graphed accumulatively and the top curve represents the percentage of runtime not spent in useful work, that is the total loss in efficiency compared to a single processor run. For 32 processors, this loss is 44%. Approximately 15% of the runtime is spent in communication overhead. While other sources (load imbalance, copy costs, duplicate computation) contribute to the parallel overhead,

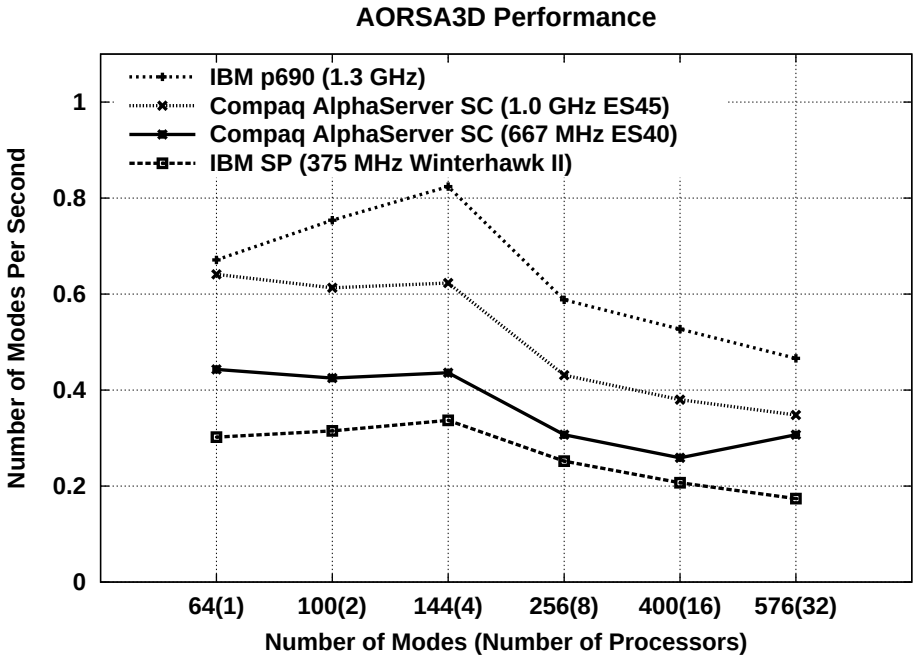

Figure 1: AORSA3D performance

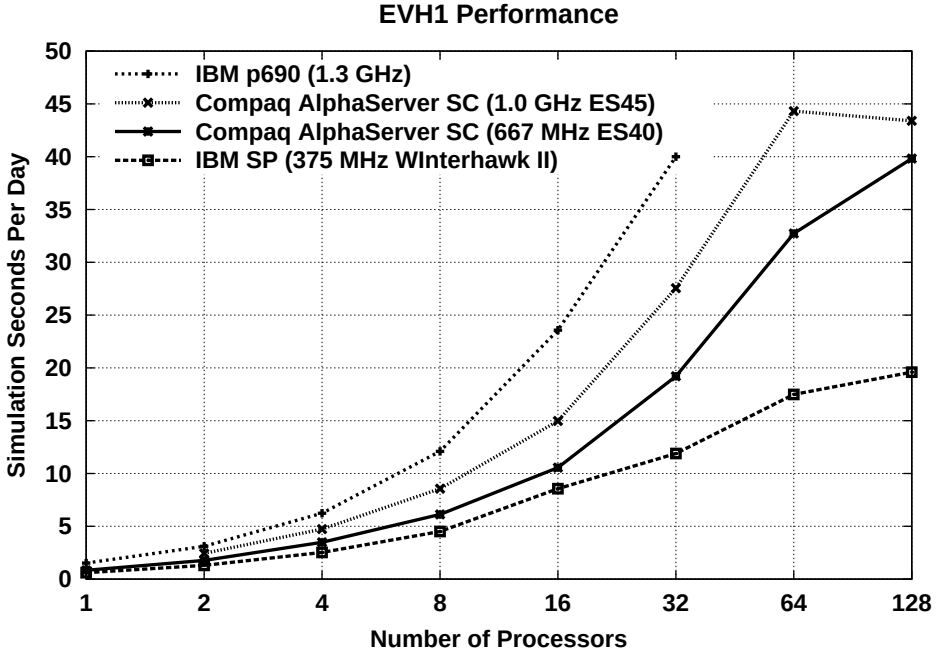

Figure 2: EVH1 performance

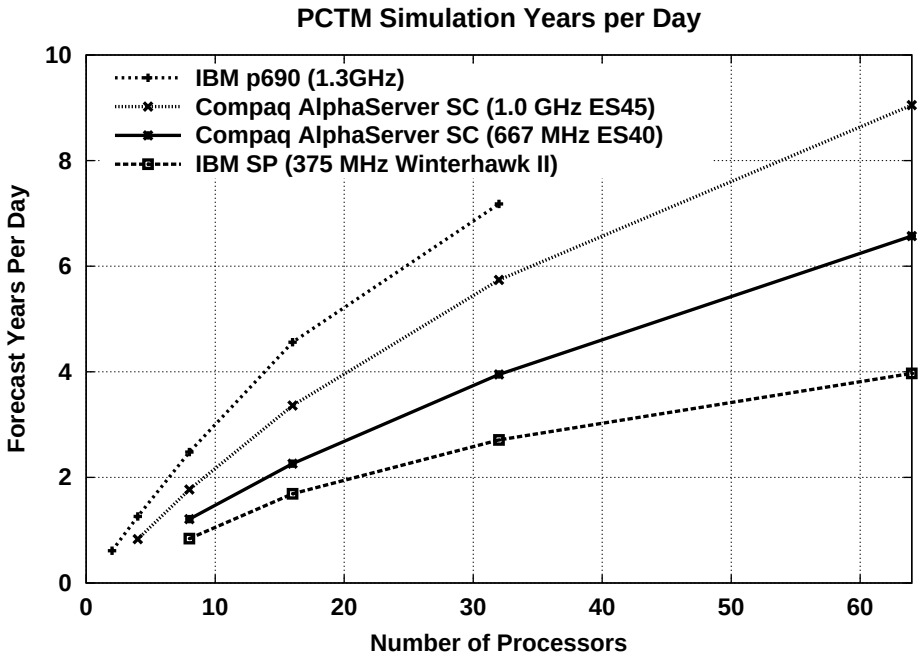

Figure 3: PCTM performance

the primary source of performance degradation for larger processor counts is not identified with any overhead category. We attribute this to a decrease in computational rate. This decrease could be due to locality issues or memory contention. It could also be due to the change in granularity from the domain decomposition, where loops getting shorter could result in less-efficient streaming. It is probably a function of all three, and we hope to diagnose the problem in the near future. Note that this particular scaling behavior is not seen in any of our other application codes, nor is it seen for CCM/MP-2D on the other systems. For example, the unidentified performance loss never accounts for more than 10% of the runtime on the Compaq systems.

It will also be interesting to measure performance when using more than 32 processors. Up to 32 processors, all processors are on the same node, so the memory demands go up even though the per-process memory granularity decreases as a function of the number of processors. Using two nodes (64 processors) will increase MPI communication costs, but it will also approximately halve the memory requirements.

Application Summary. The four application benchmarks differ significantly in performance characteristics. All four achieve at least 63% of the improvement implied by the increase in the processor clock over that of the IBM SP at ORNL, with all but CCM/MP-2D reaching at least 77% when using 32 processors. The p690 is also the best performing platform (among those listed) for all applications.

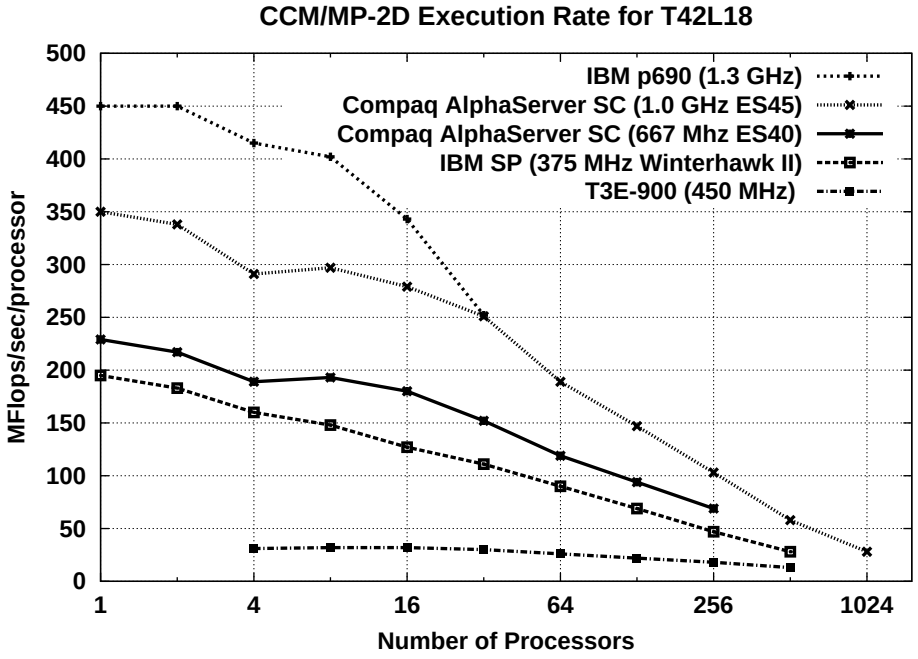

Figure 4: CCM/MP-2D performance

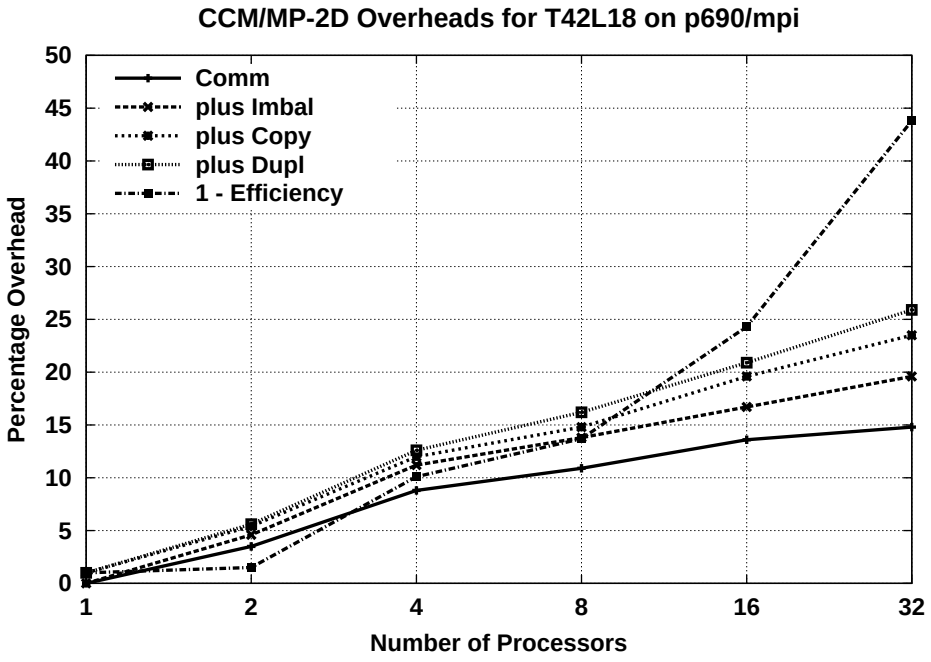

Figure 5: CCM/MP-2D performance analysis

4.2 Kernel Benchmarks

Linpack. For the LINPACK experiments, the following settings were used as input to the HPL code.

```
N      :   54000
NB     :     200
P      :       8
Q      :       4
PFACT  :   Right
NBMIN  :       8
NDIV   :       2
RFACT  :    Left
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Binary-exchange
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
```

On a single p690 node, a 32 process run achieved a computational rate of 93.5 GFlop/s with MPI only. The use of both MPI and threads also worked, but it resulted in a slight performance degradation for the single-node experiments.

This MPI-only result represents approximately 56% of peak on a 32-way node. The best observed single-processor rate was 3.3 GFlops/s, or 63% of peak. In contrast, a single POWER3-II processor in a Winterhawk-II SMP node of the ORNL SP attains roughly 85% of peak for this benchmark. The improvement of single-processor p690 performance over that of the SP is 74% of the clock ratio between the two processors.

PSTSWM. The PSTSWM benchmark experiments examine serial performance, both using one processor and running the serial benchmark on multiple processors simultaneously. Figure 6 displays the effect of increasing horizontal problem resolution on single-processor performance for 18 vertical levels. Each problem size requires approximately 4 times as much memory as the next smaller problem size. For example, T5 requires approximately 60KB of data space for each vertical level, while T85 requires approximately 18MB for each vertical level.

The improvement in single-processor performance of the p690 over that of the NERSC SP is 2.6-3.3, or 75% to 94% of the clock ratio. Like the POWER3, the POWER4 processor shows superior performance compared to the Alpha EV-series processors for codes dominated by floating-point multiply-add, such as PSTSWM. But POWER4 performance is much more sensitive to the problem size.

Figure 7 displays results for the serial benchmark run on all processors in an SMP node simultaneously. On the p690, 32 instances of the PSTSWM serial test run simultaneously. On the 4-way and 16-way SMP nodes, 4 and 16 instances of the benchmark run simultaneously, respectively.

The improvement in per-processor performance of the p690 over that of the NERSC SP ranges from 1.3 to 2.8 for this experiment, much worse than for the single-process experiment. Performance is strongly limited by memory performance when all processors are attempting to solve larger problems. The p690 memory subsystem does a poorer job maintaining processor performance than the other systems. This is an extreme case, but it does indicate that understanding the memory-subsystem characteristics is important when optimizing for the p690 architecture.

PCRM. The data structures used by PCRM are 3D arrays dimensioned as longitude $\times$ vertical $\times$ latitude. The PCRM experiment has a fixed problem size of 512 columns (fixed longitude, latitude coordinates), each with 18 vertical levels. The lengths of the longitude and latitude dimensions are varied, keeping their product fixed at 512. This experiment examines cache and vectorization (pipelining) sensitivity. It also examines the performance impact on the column physics in CCM/MP-2D of different domain-decomposition strategies.

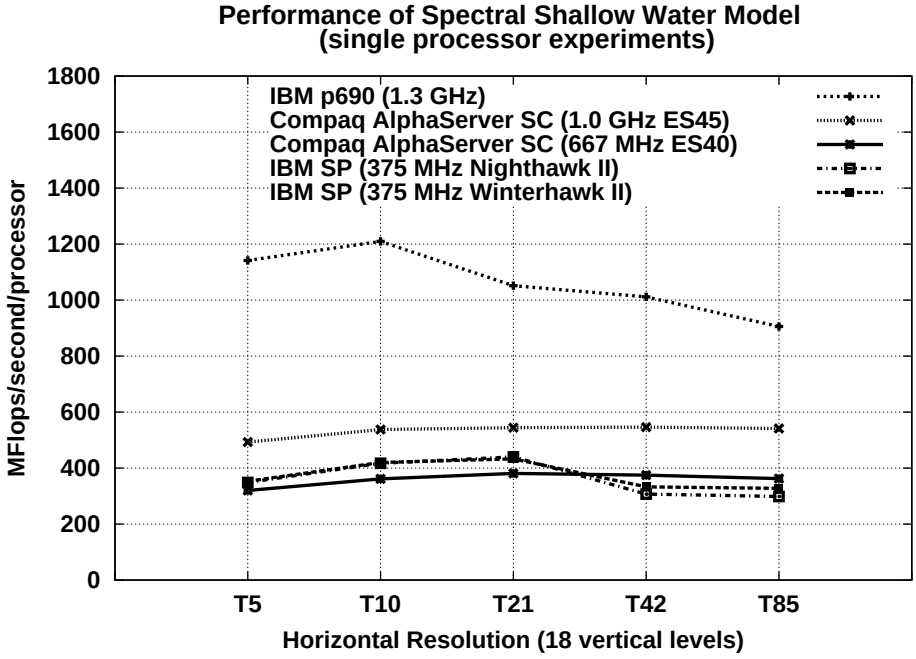

Figure 6: PSTSWM single-processor performance

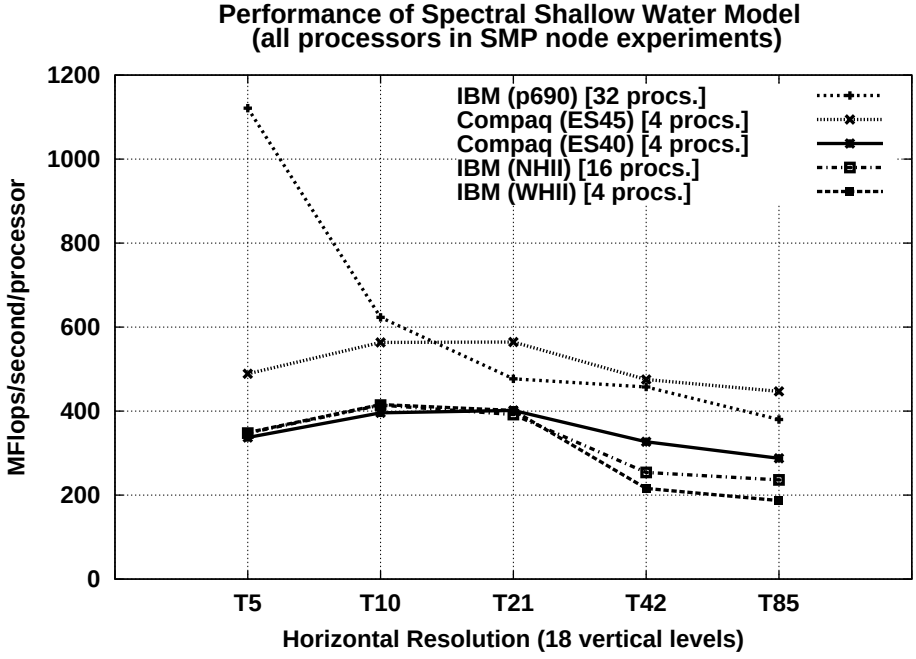

Figure 7: PSTSWM multiple-processor performance

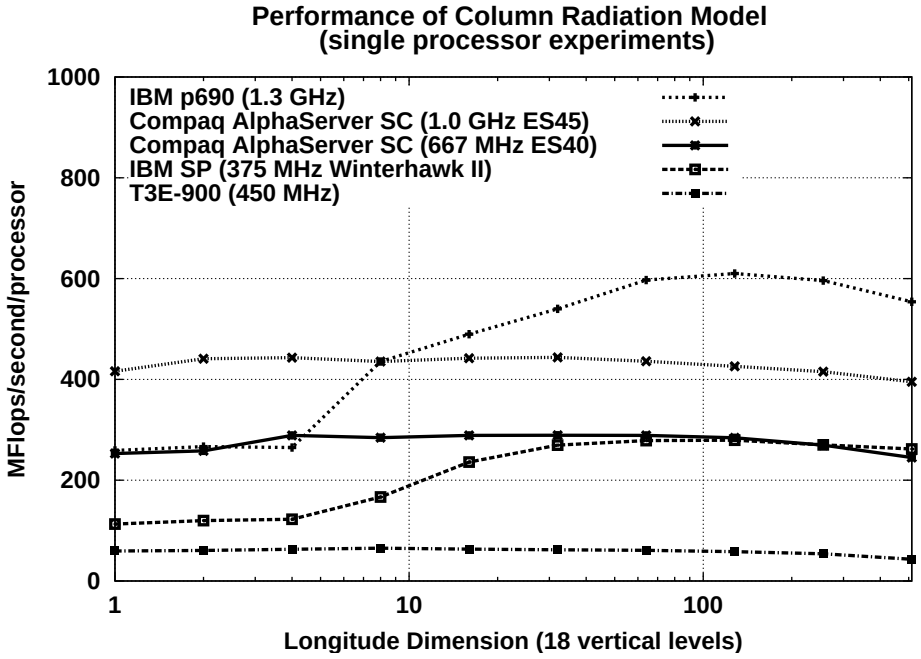

Figure 8: PCRM single-processor performance

The results in Figure 8 show that the POWER4 processor has the same qualitative performance characteristics as the POWER3-II, but with a factor of 2.0-2.6 performance improvement, representing 57%-74% of the clock ratio. IBM provides a vector intrinsic library that exhibits significantly better performance than computing scalar intrinsics such as "`sqrt`" and "`exp`" when many of them need to be computed. The longer the inner (longitude) loop, the more intrinsics the compiler can find, providing significantly improved performance. The Alpha-based systems do not show a similar performance signature.

Running this experiment using all processors in an SMP node simultaneously has little impact. Unlike PSTSWM, the performance of PCRM is only mildly sensitive to node memory bandwidth.

COMMTEST. Using COMMTEST, we measured the peak bidirectional SWAP (ping-ping) bandwidth for five experiments:

1) processor 0 swaps data with processor 1

2) processor 0 swaps data with processor 2

3) processor 0 swaps data with processor 16

4) processor i swaps data with processor $i+1$ for $i = 0, 2, \ldots, P-1$. For the p690, 16 pairs of processors (0-1, 2-3, ..., 14-15) swap data simultaneously.

5) processor i swaps data with processor $i+P/2$ for $i = 0, \ldots, P/2-1$. For the p690, 16 pairs of processors (0-16, 1-17, ..., 15-31) swap data simultaneously.

We measured SWAP bandwidth because the swap operator is more commonly used in performance critical interprocessor communication in our application codes than is the unidirectional send (receive). Note that processes were explicitly bound to processors in these experiments to ensure that the indicated message-passing patterns did in fact occur.

The COMMTEST results are displayed in Figure 9 using both log-log and log-linear plots. Performance for small messages (length less than 1 KByte) is similar across all experiments. For larger messages, performance differs dramatically, reflecting different demands on the memory subsystem. Swapping data between

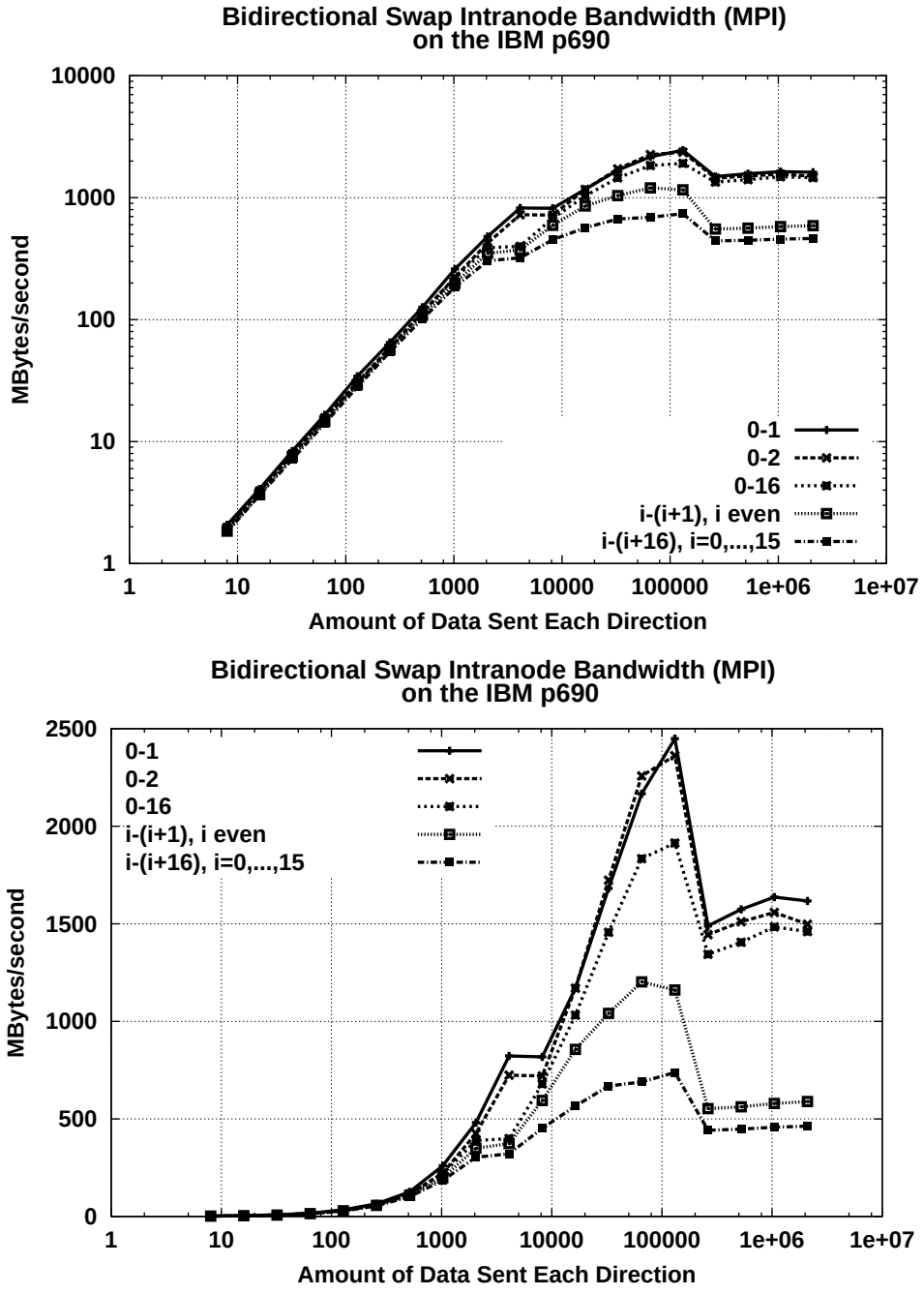

Figure 9: Log-log and log-linear plots of MPI intranode bandwidth

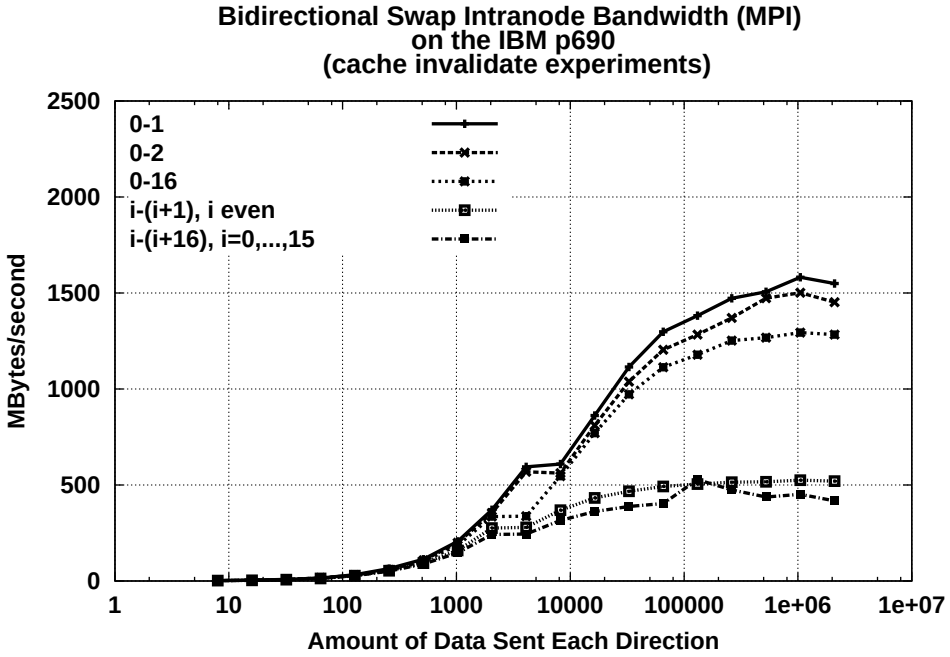

Figure 10: MPI intranode bandwidth when first invalidating the L2 cache

processors sharing an L2 cache almost always exhibits the highest performance. Swapping data between processors within the same MCM, but not sharing an L2 cache, is slightly slower. Remember that processors within the same MCM can snoop each others L2 caches.

Swapping data between processors in different MCMs is significantly slower for certain ranges of message sizes. For the largest message sizes, the difference between the different single pair experiments is relatively small. Simultaneous swapping of messages between 16 pairs of neighboring processors is significantly slower than the single pair experiments, and the simultaneous swapping of messages between 16 pairs of "distant" processors is the slowest of the five experiments.

The drop in performance when swapping messages of size 256 KBytes or larger is apparently due to an increase in L2 cache misses. When explicitly invalidating the L2 cache before each experiment, as shown in Figure 10, the performance curves change. Now the bandwidth increases essentially monotonically for all message sizes and achieves nearly the same bandwidth rate for 2 MByte messages as when not invalidating the L2 cache. The performance differences between experiments 1, 2, and 3 are somewhat smaller, as are the performance differences between experiments 4 and 5, but are qualitatively the same. Which performance rates will be seen in practice (with or without cache invalidation) is application dependent.

Kernel Summary. Like the application benchmarks, the three computational kernel benchmarks differ significantly in performance characteristics. When using a single processor all three achieve at least 60% of the improvement implied by the increase in the processor clock over that of the IBM SP. The multiple-processor runs indicate a potential for performance degradation due to memory (or other) contention. The MPI communication tests also indicate a performance sensitivity to the memory hierarchy. The next set of experiments look more carefully at issues of memory contention and process placement.

4.3 Process-Binding Experiments

The process-binding experiments examine the performance impact of using only some of the processors in the p690 node and of controlling how processes are assigned to processors. This allows us to assess the performance of these codes on a p690 HPC system, which has 16 processors per SMP node and in which

processors do not share an L2 cache. It also permits us to examine more closely the source of performance degradation identified in the previous benchmarking results.

A set of tools (scripts and libraries) were provided by Farid Parpia of IBM for binding processes and threads to processors. The first experiment examines whether the performance degradation in the multiple-processor PSTSWM experiments is due to memory contention or interactions with system processes.

Figure 11 shows the performance effect of increasing the horizontal problem resolution when 1, 8, 16, 24, 30, or 32 processors are running the experiment simultaneously. For the larger horizontal resolutions (T21, T42, and T85), the performance degradation appears to be a relatively smooth function of the number of processes. This suggests that the degradation is primarily a function of L3 or main-memory bandwidth limitations. For the smaller horizontal resolutions (T5 and T10), the problem size fits more easily into the L2 cache. We made no attempt to control the assignment of processes in these experiments, but it appears that the processes in the 8 and 16 process experiments were assigned to processors that did not share L2 caches. This was not possible for the 24, 30 and, 32 process experiments, and the T10 performance likely suffers from a significant increases in L2 cache misses for these processor counts due to this forced sharing.

The next experiments examine the effect of controlling process binding. Figure 12 shows the results from three 16-processor runs:

- not binding processes (default);
- bind processes to every other processor, starting with processor 0 (alternating);
- bind processes to consecutive processors, starting with processor 0 (consecutive).

The alternating binding guarantees that no processes will share an L2 cache, and it spreads the processes across more MCMs and thus more L3 caches. The consecutive binding guarantees that all processes reside in the first two 8-processor MCMs, maximizing L2-cache sharing.

From these results, it is clear that assigning processes to consecutive processors can significantly decrease the performance of this benchmark. In the most extreme case (T10), the alternating binding policy is almost 100% better than the consecutive binding policy. Repeating the experiment using only 8 processors produces similar results. Comparing the 8- and 16-process results suggests that the performance degradation for the smaller problem sizes is probably due to the L2-cache sharing. For the larger problem sizes, the L2-cache sharing and the number of MCMs involved in the experiment are both important in determing performance. Note that the performance of the "default" process assignment is almost identical to that of the alternating binding policy, which is itself similar to what would be achieved on an HPC version of the p690.

PSTSWM represents a memory "torture test", with performance strongly dependent on the performance of the memory subsystem. The next experiments examine whether similar effects are observed with the application codes. CCM/MP-2D demonstrates a large, as yet unidentified, performance degradation when using all 32 processors. Figure 13 describes the scaling performance of CCM/MP-2D when using the alternating, default, and consecutive process assignment options. We also include performance data for two more binding policies:

- bind processes to every fourth processor, starting with processor 0 (alternating4), and
- bind processes to every eighth processor, starting with processor 0 (alternating8).

The alternating binding is 8%-9% better than the consecutive binding for 2 and 4 processes, and 22%-25% better for 8 and 16 processes. For up to 4 processes, the alternating and consecutive bindings are on a single MCM and differ only in the sharing of the L2 cache. By comparison with the other binding policies, additional performance is gained from also spreading the processes over multiple MCMs. For example, performance for the default binding policy is 15%–18% better than that for the consecutive binding policy for 2 and 4 processes, and 30% better for 8 processes. Note that the default process assignment appears to maximize the distance between processors, achieving the same performance as alternating8 for 2 and 4 processes, nearly the same as alternating4 for 8 processes, and the same as alternating for 16 processes. The binding itself appears to have no significant performance impact, postive or negative.

Figure 14 shows the overhead analysis for the CCM/MP-2D default, consecutive, and alternating binding policies. The communication, duplicate computation, and copy overheads are all nearly identical for the three

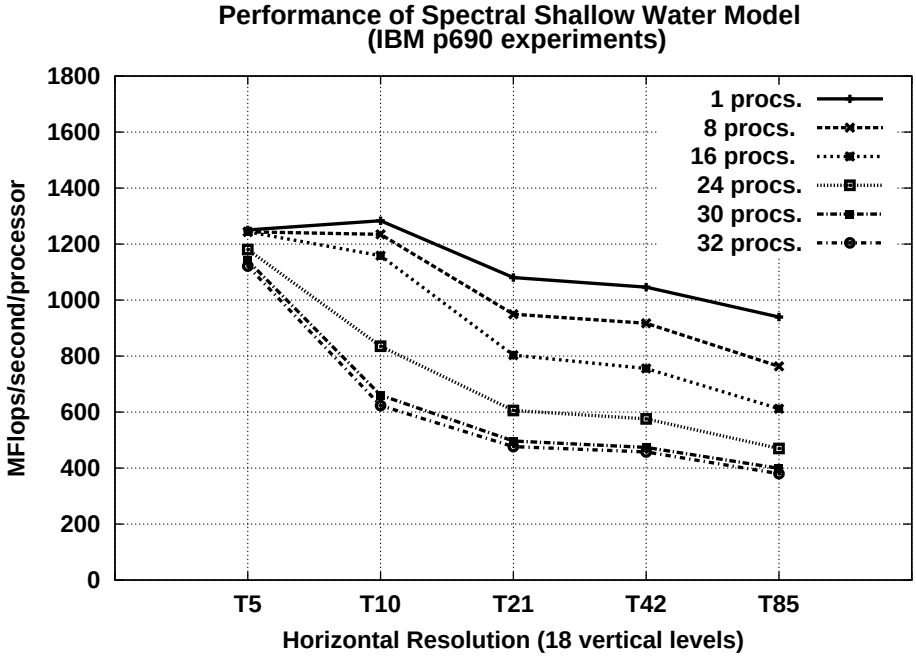

Figure 11: PSTSWM processor-scaling experiments

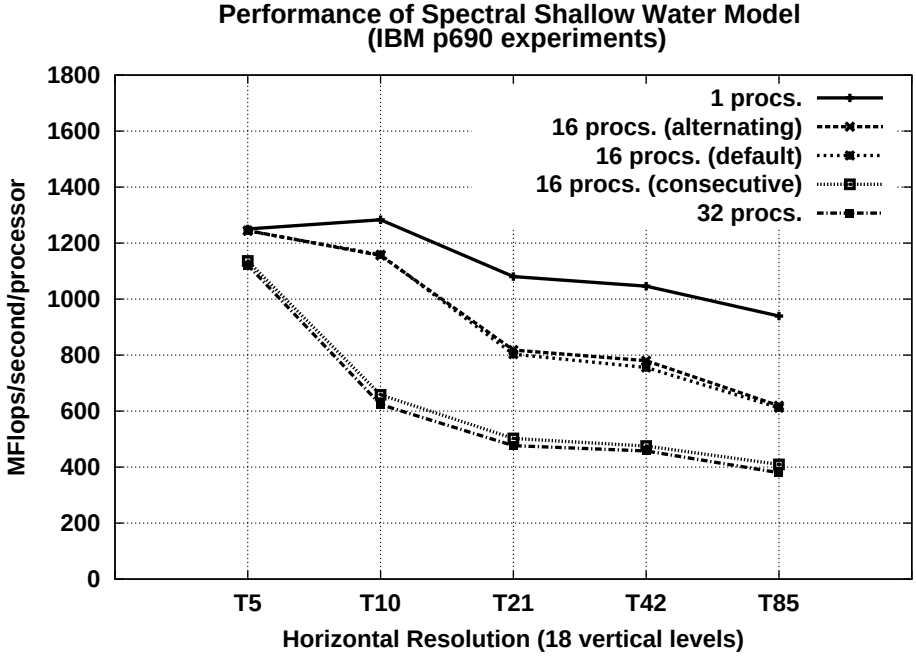

Figure 12: PSTSWM process-binding experiments

policies, and do not appear in the figure. The metrics that vary are load imbalance and the "other" categories, both reflecting changes in serial computational rates. Remember that the alternating and consecutive binding policies are both restricted to a single MCM when using less than 8 processors, while the default policy is unrestricted.

From these results, the CCM/MP-2D performance scaling problem appears to be explained by the memory performance. When using the consecutive binding policy, the unidentified overhead grows relatively smoothly. It is only with the default process assignment that it suddenly becomes an issue when using 16 and 32 processes.

Our hypothesis is that the differences between the consecutive and alternating policies for small process counts are dominated by increased L2-cache misses caused by L2-cache sharing. The differences between the default and alternating policies for less than 16 processes are dominated by the superior system memory bandwidth that the default policy can achieve because it is not restricted to one or two MCMs.

Figure 15 describes the equivalent binding experiments for PCTM. In contrast to CCM/MP-2D, PCTM performance scalability on the p690 is similar to that on the other platforms. The advantage of alternating binding compared to consecutive binding for PCTM is 4%-6% for 2 and 4 processes and 27%-30% for 8 and 16 processes. The advantage of default assignment compared to consecutive binding is 15%-23% for 2 and 4 processes, and 40% for 8 processes. These results are qualitatively the same as those for the CCM/MP-2D experiments. For both codes it appears that L2-cache sharing has less performance impact than overall L3 and main-memory bandwidth. Unlike for CCM/MP-2D, process binding improved performance slightly for the 32-process PCTM runs.

While process binding was important in these experiments, allowing us to examine performance characteristics of the p690, it does not appear to be useful for production runs of MPI-only applications. Early experiments with mixed MPI-OpenMP applications show that binding in consecutive fashion has some performance benefit, probably due to L2 cache reuse among threads. Also note that the experiments utilizing less than 32 processes represent a real use of a p690 node. Applications with large memory requirements may only fit 16 or fewer MPI tasks to a node.

4.4 Node-Sharing Experiments

Our final experiments examine the impact of running multiple jobs on a single node. The issue is whether it is better to run multiple jobs each using less than 32 processors or one job using all 32 processors. Throughput is often enhanced by running multiple "small" parallel jobs rather than one single large parallel job, because the small jobs have higher parallel efficiencies. However, if the same problem is being solved in each case, then the multiple small parallel jobs require significantly more memory than the single large parallel job. The performance impact of the increased memory usage could offset the improved parallel efficiency.

In these experiments we compare the throughput for running eight 4-process jobs, four 8-process jobs, two 16-process jobs, or one 32-process job. We also consider three different binding strategies, as follow.

- Interleave the processors assigned to different jobs. For example, if running J jobs, then process i of job j is bound to processor $i * J + j$. (interleaved)
- Do not bind processes. (default)
- Bind processes to consecutive processors. For example, if running J jobs, then process i of job j is bound to processor $(j-1) * J + i$. (block)

Figure 16 describes the results of the node-sharing experiment for PCTM. These data show a 6% performance degradation from running four 8-way jobs instead of two 16-way jobs, and a 7% improvement when running two 16-way jobs instead of one 32-way job. The two 16-way jobs suffer a 27% degradation in per job performance compared to a single 16-way job, and the four 8-way jobs suffer a 29% degradation in per job performance compared to a single 8-way job. In contrast, the parallel efficiencies for the 8-way and 16-way PCTM runs are almost identical, and the parallel efficiency of the 32-way job is 29% less than that of the 16-way job. Thus, the performance degradation when running multiple jobs is comparable to the loss in efficency when using all of the processors for a single parallel job.

Figure 17 describes the results of the node-sharing experiment for CCM/MP-2D. These data show a 19%

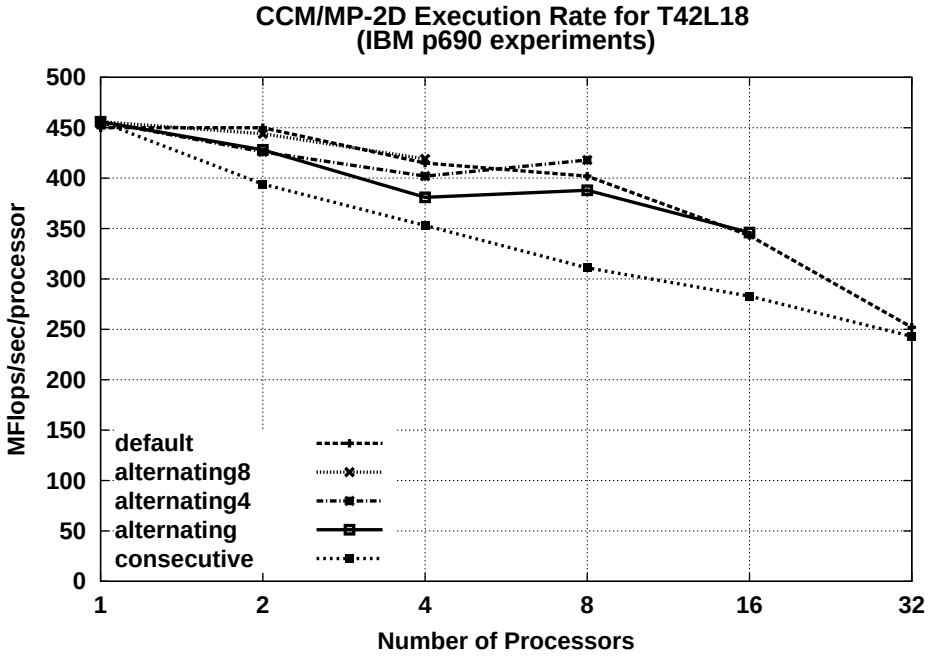

Figure 13: CCM/MP-2D process-binding experiments

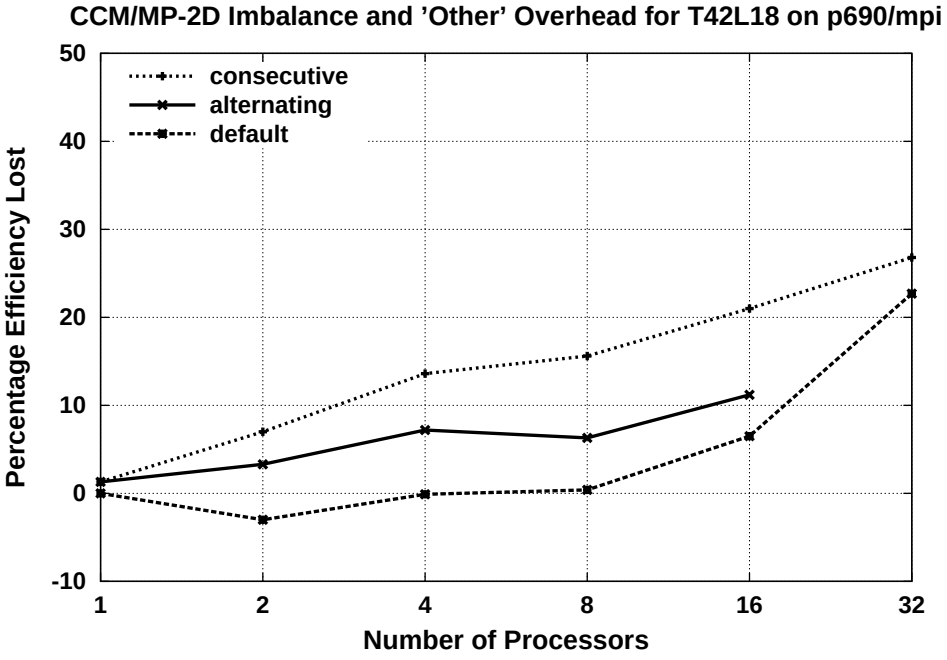

Figure 14: CCM/MP-2D process-binding experiments

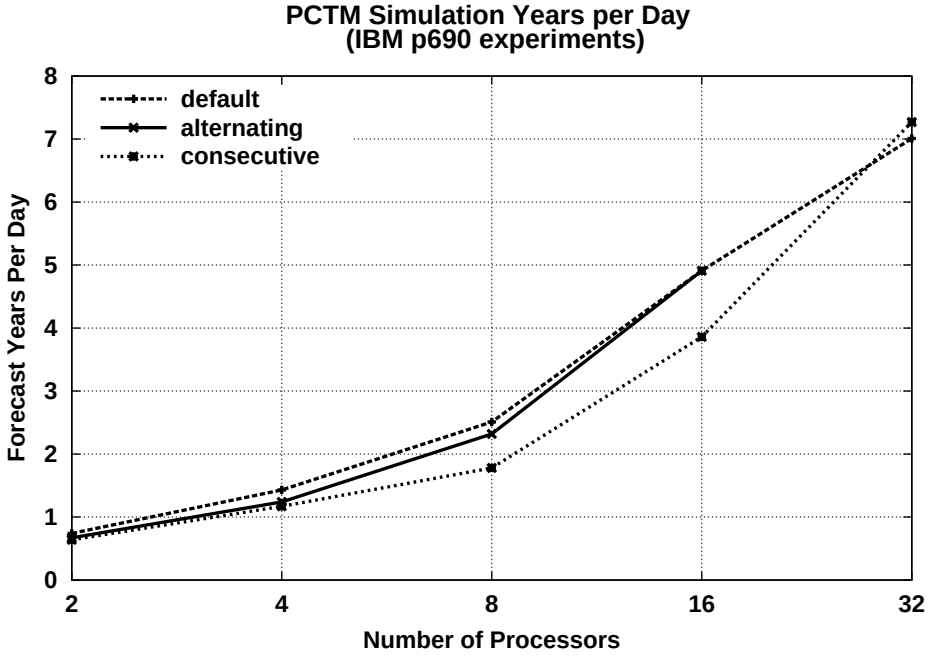

Figure 15: PCTM process-binding experiments

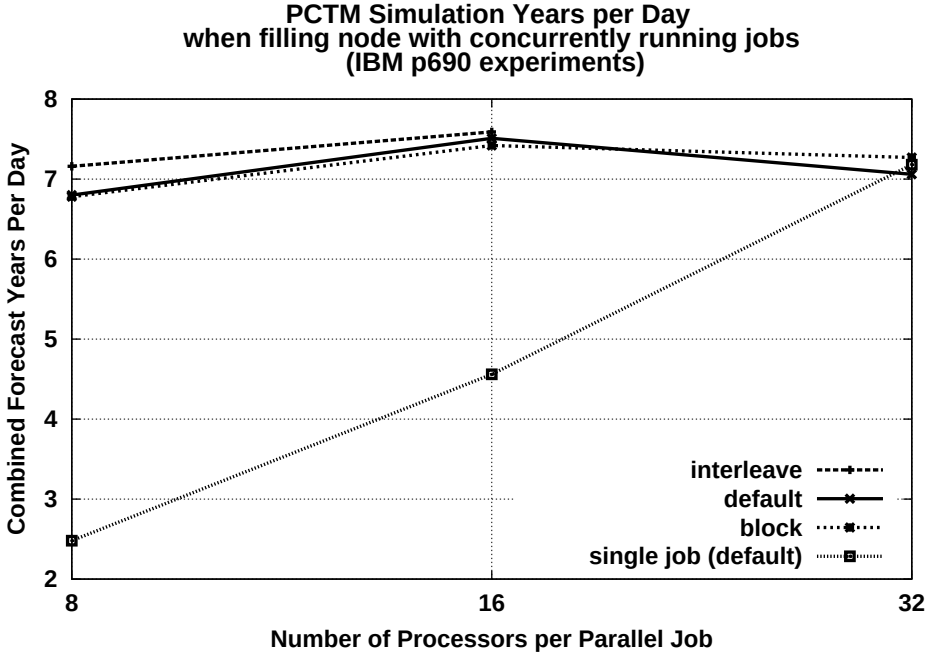

Figure 16: PCTM node-sharing experiments

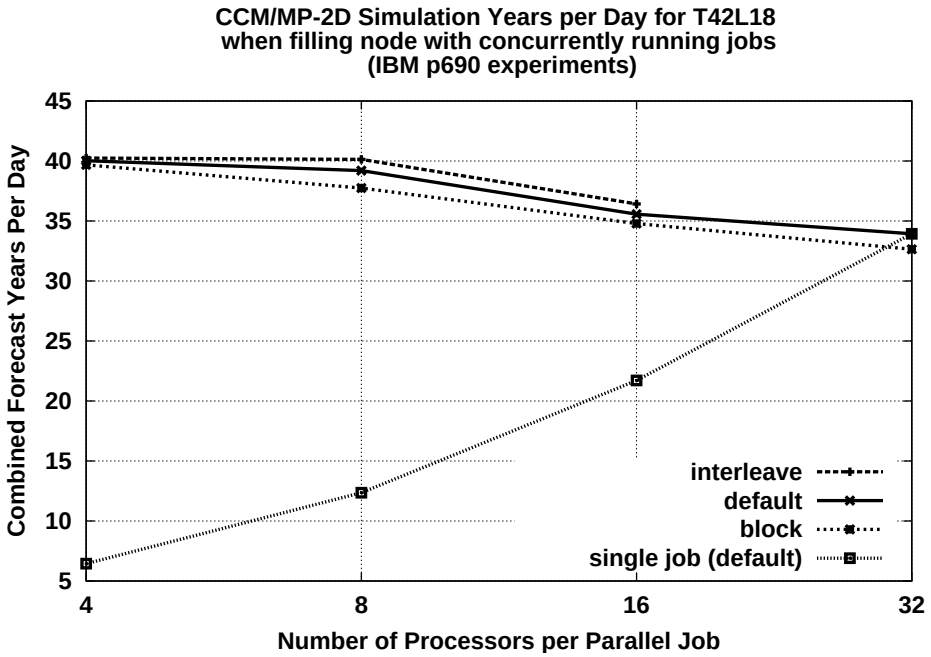

Figure 17: CCM/MP-2D node-sharing experiments

improvement in throughput from running either eight 4-way or four 8-way T42L18 jobs rather than one 32-way job. The 4-way, 8-way, and 16-way multiple job runs demonstrate 24%, 21%, and 16% degradation in per job performance compared to the corresponding single job performance, respectively. However, this is not as much as the 30% loss in efficiency when using 32 processors instead of 8 processors for a single job. Note that the binding policies did not have a significant performance impact, although the interleave binding was always the best.

Results from the two codes lead to somewhat different conclusions about the efficacy of node-sharing. In general, both the parallel efficiency of the 32-way job and the memory requirements of the smaller parallel jobs determine whether node-sharing can be used to improve throughput for a given application and problem size on the p690.

5 Conclusions

As mentioned earlier, porting the pure MPI codes used in this study from a POWER3 system to the IBM p690 was straightforward. This is the first, and possibly most important, result of our evaluation.

The second result is that the p690 is at least twice as fast as the previous generation of IBM high-performance system across all our benchmarks. While the peak performance is 3.5 times that of the previous systems, peak represents only processor performance. The memory subsystem of the SMP node is at least as important.

The third result is that the performance of the p690 memory subsystem is not as balanced with processor performance as in the previous IBM (and Compaq) systems. While L2-cache sharing is at least partly to blame for some of the performance loss, overall node bandwidth can also be a performance bottleneck. The importance of the memory subsystem in obtaining good performance also makes it difficult to determine a priori the utility of using node-sharing to improve throughput.

IBM has proposed a number of techniques for ameliorating the node bandwidth issue. For example, the POWER4 processor has memory-prefetch streams to hide latency, but these streams must start on memory-page boundaries. The relatively small AIX page size of 4KB limits the effectiveness of POWER4 streaming. Support for 16MB pages has recently become available. Such “large pages” may substantially

enhance the performance of large-memory applications that take advantage of prefetching. We have just begun examining performance on systems with large pages. Our initial evaluation results indicate significant performance improvement for the PSTSWM benchmark, but no performance improvement on any of the full application codes. We are continuing to investigate.

6 Acknowledgements

We gratefully acknowledge the following institutions for access to their HPC systems:

- the Pittsburgh Supercomputing Center for access to the Compaq AlphaServer SC at PSC,
- the Center for Computational Sciences at ORNL for access to the IBM p690, IBM SP, and Compaq AlphaServer SC at ORNL, and
- the National Energy Research Scientific Computing Center for access to the IBM SP and the Cray Research T3E at NERSC,

and the aid and encouragement of the following computational scientists and developers:

- Warren Washington and Tom Bettge at the National Center for Atmospheric Research for specifying and providing the PCTM benchmark,
- John Blondin at North Carolina State University, Tony Mezzacappa at ORNL, and William (Raph) Hix at ORNL for specifying and providing the EVH1 benchmark,
- Don Batchelor and Fred Jaeger at ORNL for specifying and providing the AORSA3D benchmark, and
- Vance Shaffer and Farid Parpia of IBM for supplying the process binding scripts and libraries.

References

[1] T. W. Bettge, A. P. Craig, R. James, V. B. Wayland, and W. G. Strand, *The DOE Parallel Climate Model (PCM): Computational highways and backroads*, in International Conference on Computational Science, V. N. Alexandrov, J. Dongarra, B. A. Juliano, R. S. Renner, and C. J. K. Tan, eds., vol. 2073 of Lecture Notes in Computer Science, New York, 2001, Springer, pp. 149–158.

[2] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[3] J. M. Blondin and E. Lufkin, *Piecewise parabolic method for curvilinear coordinates*, Astrophysical Journal Supplement Series, 88 (1993), pp. 589–594.

[4] P. Colella and P. R. Woodward, *The Piecewise Parabolic Method (PPM) for gas-dynamical simulations*, J. Comp. Phys., 54 (1984), pp. 174–201.

[5] J. Dongarra, *Performance of various computers using standard linear equations software*, Computer Science Technical Report CS-89-85, University of Tennessee, Knoxville TN, 37996, July 2002. (see also http://www.netlib.org/benchmark/hpl).

[6] J. B. Drake, I. T. Foster, J. G. Michalakes, B. Toonen, and P. H. Worley, *Design and performance of a scalable parallel community climate model*, Parallel Computing, 21 (1995), pp. 1571–1591.

[7] E. F. Jaeger, L. A. Berry, E. D'Azevedo, D. B. Batchelor, , and M. D. Carter, *Advances in full-wave modeling of radio frequency heated, multidimensional plasmas*, Phys. Plasmas, 9 (2002), pp. 1873–1881.

[8] P. W. Jones, *The Los Alamos Parallel Ocean Program (POP) and coupled model on MPP and clustered SMP computers*, in Making its Mark – The Use of Parallel Processors in Meteorology: Proceedings of the Seventh ECMWF Workshop on Use of Parallel Processors in Meteorology, G.-R. Hoffman and N. Kreitz, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1999.

[9] J. T. Kiehl, J. J. Hack, G. Bonan, B. A. Boville, D. L. Williamson, and P. J. Rasch, *The National Center for Atmospheric Research Community Climate Model: CCM3*, J. Climate, 11 (1998), pp. 1131–1149.

[10] Office of Science, U.S. Department of Energy, *Scientfic Discovery Through Advanced Computing*. (available from http://www.sc.doe.gov/ascr/mics/scidac/SciDAC_strategy.pdf), March 2000.

[11] W. M. Washington, J. W. Weatherly, G. A. Meehl, J. A. J. Semtner, T. W. Bettge, A. P. Craig, W. G. Strand, J. M. Arblaster, V. B. Wayland, R. James, and Y. Zhang, *Parallel Climate Model (PCM) control and transient simulations*, Climate Dynamics, 16 (2000), pp. 755–774.

[12] P. H. Worley and I. T. Foster, *PSTSWM: a parallel algorithm testbed and benchmark code for spectral general circulation models*, Tech. Rep. ORNL/TM–12393, Oak Ridge National Laboratory, Oak Ridge, TN, (in preparation).

[13] P. H. Worley and B. Toonen, *A users' guide to PSTSWM*, Tech. Rep. ORNL/TM–12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.